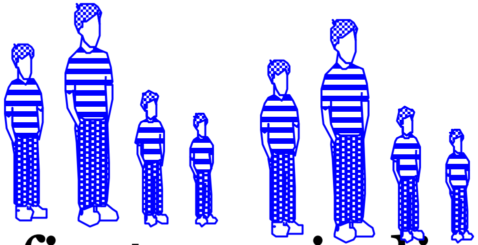
Queue

Bruce Nan

Queue

- Stores a set of elements in a particular order
- Queue principle: FIRST IN FIRST OUT
- = FIFO
- It means: the first element inserted is the first one to be removed
- Example

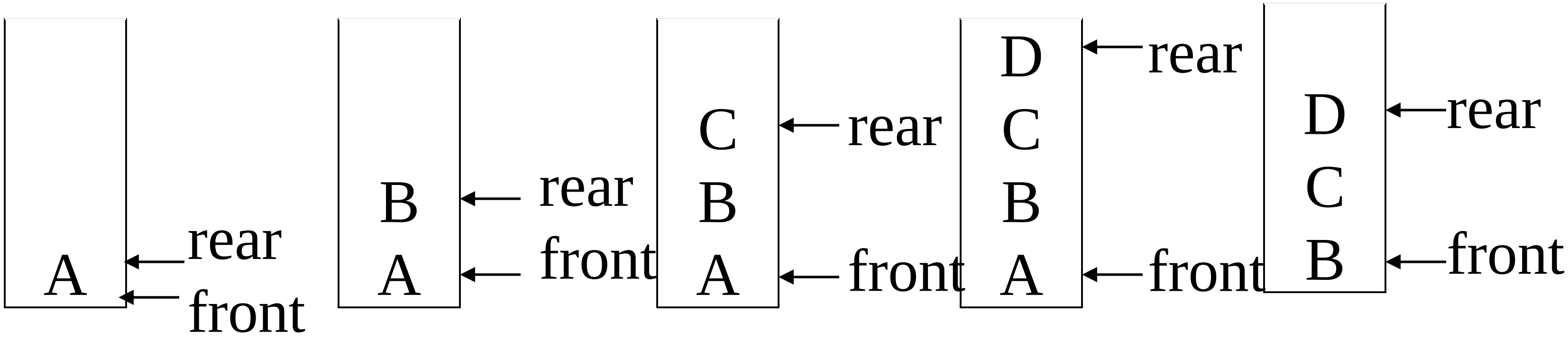


- The first one in line is the first one to be served

Queue Applications

- Real life examples
  - Waiting in line
  - Waiting on hold for tech support
- Applications related to Computer Science
  - Threads
  - Job scheduling

First In First Out



Applications: Job Scheduling

front	rear	Q[0]	Q[1]	Q[2]	Q[3]	Comments
-1	-1					queue is empty
-1	0	J1				Job 1 is added
-1	1	J1	J2			Job 2 is added
-1	2	J1	J2	J3		Job 3 is added
0	2		J2	J3		Job 1 is deleted
1	2			J3		Job 2 is deleted

Queue ADT

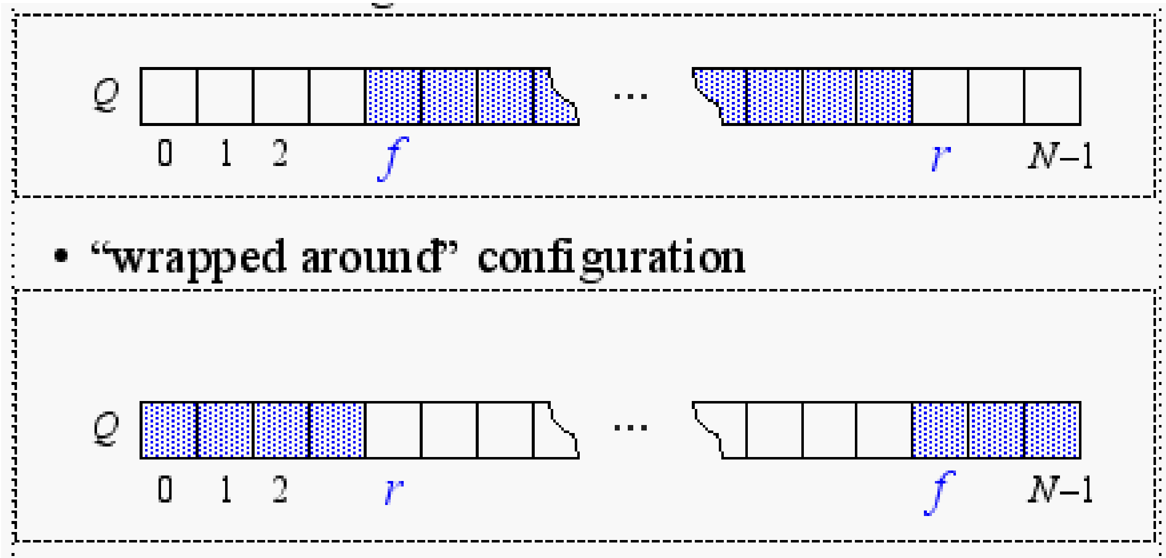
```
objects: a finite ordered list with zero or more elements.
methods:
  for all queue ∊ Queue, item ∊ element,
          max_ queue_ size ∊ positive integer
  Queue createQ(max_queue_size) ::=
          create an empty queue whose maximum size is
          max_queue_size
  Boolean isFullQ(queue, max_queue_size) ::=
          if(number of elements in queue == max_queue_size)
          return TRUE
          else return FALSE
  Queue Enqueue(queue, item) ::=
          if (IsFullQ(queue)) queue_full
          else insert item at rear of queue and return queue
```

Queue ADT (cont'd)

```
Boolean isEmptyQ(queue) ::=
        if (rear - front ==0)
        return TRUE
        else return FALSE
Element dequeue(queue) ::=
        if (IsEmptyQ(queue)) return
         else remove and return the item at front of queue.
```

Array-based Queue Implementation

- As with the array-based stack implementation, the array is of fixed size
  - A queue of maximum N elements
- Slightly more complicated
  - Need to maintain track of both front and rear



Implementation 1

Implementation 2

Implementation 1: createQ, isEmptyQ, isFullQ

```
Queue createQ(max_queue_size) ::=
# define MAX_QUEUE_SIZE 100/* Maximum queue size */
typedef struct {
            int key;
            /* other fields */
            } element;
element queue[MAX_QUEUE_SIZE];
int rear = -1;
int front = -1;
Boolean isEmpty(queue) ::= front == rear
Boolean isFullQ(queue) ::= rear == MAX_QUEUE_SIZE-1
```

Implementation 1: enqueue

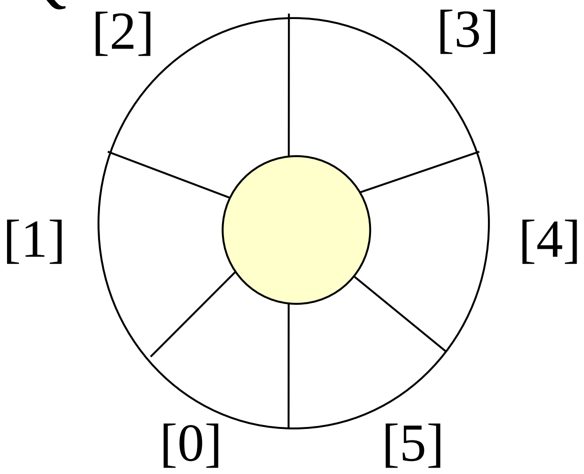
```
void enqueue(int *rear, element item)
{
/* add an item to the queue */
   if (*rear == MAX_QUEUE_SIZE - 1) {
      queue_full( );
      return;
  }
  queue [++*rear] = item;
}
```

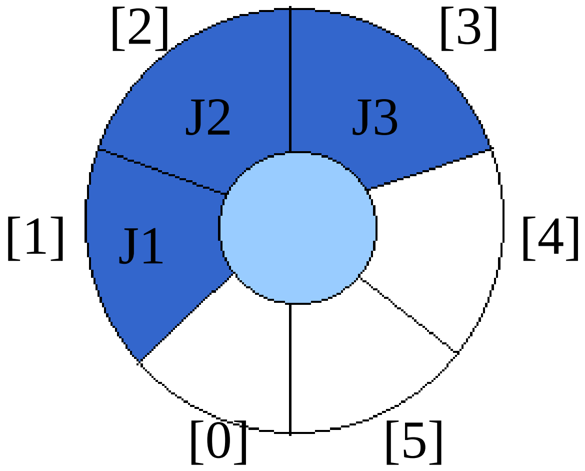
Implementation 1: dequeue

```
element dequeue(int *front, int rear)
{
/* remove element at the front of the queue */
  if ( *front == rear)
    return queue_empty( );     /* return an error key */
  return queue [++ *front];
}
```

Implementation 2: Wrapped Configuration

EMPTY QUEUE





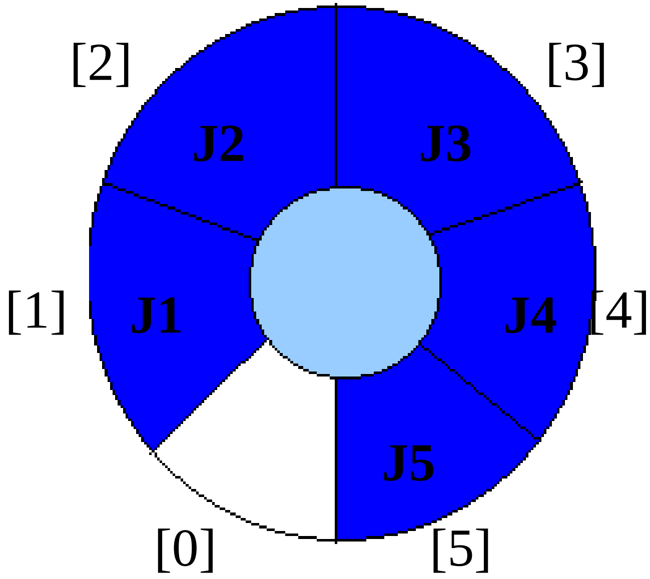
front = 0
rear = 0

front = 0
rear = 3

Can be seen as a circular queue

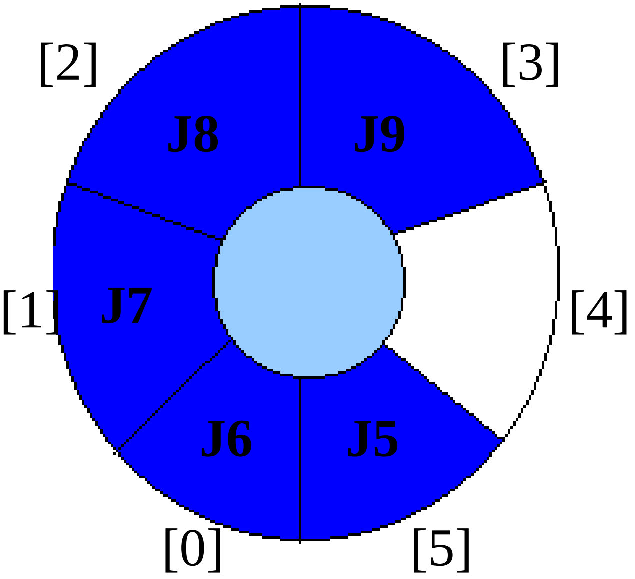
Leave one empty space when queue is full
Why?

FULL QUEUE



front =0
rear = 5

FULL QUEUE



front =4
rear =3

How to test when queue is empty?
How to test when queue is full?

Enqueue in a Circular Queue

```
void enqueue(int front, int *rear, element item)
{
/* add an item to the queue */
    *rear = (*rear +1) % MAX_QUEUE_SIZE;
     if (front == *rear) /* reset rear and print error */
     return;
   }
     queue[*rear] = item;
}
```

Dequeue from Circular Queue

```
element dequeue(int* front, int rear)
{
  element item;
  /* remove front element from the queue and put it in item */
     if (*front == rear)
        return queue_empty( );
                 /* queue_empty returns an error key */
     *front = (*front+1) % MAX_QUEUE_SIZE;
     return queue[*front];
}
```

Queue in C++

```
#include <queue>
```

```
queue<int>  int_qu;
```

empty()	Test whether queue is empty;
size()	Return queue size;
front()	Access front element;
back()	Access last element;
push()	Insert element;
pop()	Delete next element;

Queue in Java

A Queue is a collection for holding elements prior to processing.

Queue<Integer> Q = new LinkedList<>();

add(E o) Inserts the specified element into this queue

peek() Retrieves, but does not remove, the head of this queue, returning null if this queue is empty.

poll() Retrieves and removes the head of this queue, or null if this queue is empty.

Deque

- A deque is a double-ended queue
- Insertions *and* deletions can occur at *either* end
- Implementation is similar to that for queues

Deque in C++

```
#include <queue>
deque<int>  int_dq;
```

empty()	Test whether container is empty
size()	Return size;
front()	Access front element;
back()	Access last element;
push_front()	Insert element at the beginning
push_back()	Insert element at the end
pop_front()	Delete first element
pop_front()	Delete last element

Deque in Java

```
Deque<Integer> dq = new LinkedList<>();
```

offerFirst	Inserts the specified element at the front of this deque
offerLast	Inserts the specified element at the end of this deque
peekFirst	Retrieves, but does not remove, the first element of this deque
peekLast	Retrieves, but does not remove, the last element of this deque
pollFirst	Retrieves and removes the first element of this deque
pollLast	Retrieves and removes the last element of this deque